ARTICLE
REPRINT

AR-114

December 1979

Implement Complex Analog Filters with a Real-Time Signal Processor

Robert E. Holm
Product Marketing Manager
Telecommunications

Implement complex analog filters with a real-time signal processor

Traditional filter-design techniques, when applied to precision applications, often yield circuits that are difficult to reproduce consistently. The analog μP can eliminate these problems, but it requires a new analog+digital synthesization approach.

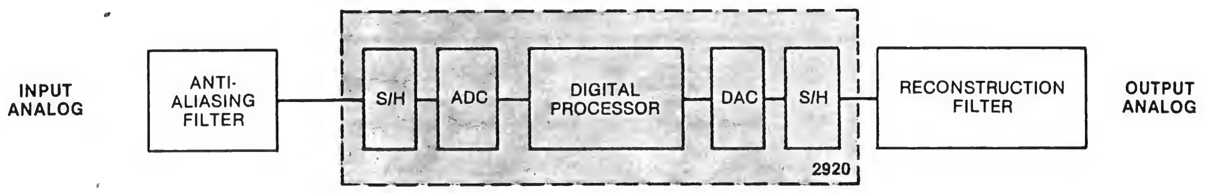
Robert E Holm, Intel Corp

Because the 2920 NMOS VLSI signal-processor IC is an analog-in, analog-out, digital-in-between device (EDN, September 20, pg 142), it helps solve several common analog-design problems. In particular, it eliminates the need for component matching and/or tuning because performance doesn't vary from device to device (digital processing is stable, predictable and repeatable). Moreover, it minimizes performance degradation arising from circuit interaction or noise.

However, the 2920 does challenge an analog designer to learn new skills—to become adept in the use of a μP development system and to learn the digital algorithms that implement analog functions. Using the device to create analog filters, for example, represents a particularly interesting—and potentially very rewarding—design challenge. This article explains the general method, then illustrates its use by means of a sophisticated application problem. First, however, a review of how the 2920 operates, as well as a look at the digital-processing fundamentals and the basic theory of digital filters, is necessary.

A sampled-data system on a chip

The 2920 signal processor can be considered a single-chip system, because it can perform all the functions illustrated in **Fig 1.** In addition, it can multiplex its input and output lines, thus enabling it to implement either several circuits or one circuit with



ANTIALIASING FILTER — Bandlimits input signal to reduce distortion due to sampling.
S/H — Sample and hold amplifier performs sampling process and holds data sufficiently long for processing.
ADC — Analog-to-digital conversion; generates a digital word to represent held analog voltage.
DIGITAL PROCESSOR — Implements transfer function using digital processing (under software control).
DAC — Converts digital words to analog voltages.
RECONSTRUCTION FILTER — Smooths D/A or S/H waveforms to recover continuous analog output signal.

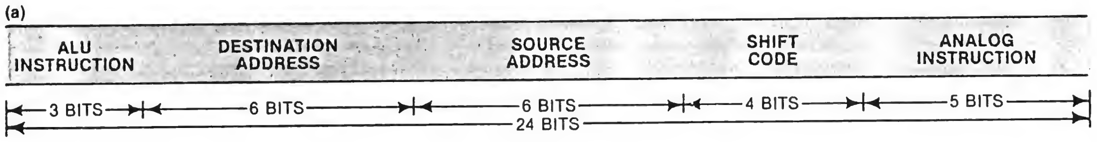
Fig 1—A general purpose sampled-data system *using a digital signal processor contains these basic blocks. Although this configuration assumes that both the input and output signals are analog, this isn't a necessary condition; digital signals can be considered a special type of analog signal and thus processed accordingly.*

digital sections—stores these bits. During A/D conversion, the DAR accumulates the digital word until conversion is complete; this word is then loaded into a scratchpad-RAM location for further processing. When the IC outputs a value, it loads the nine most significant bits of a RAM location into the DAR. The DAR drives the chip's D/A converter, whose output can be routed to any of eight analog outputs by the output demultiplexer and sample/hold amplifiers.

While these analog operations are taking place, the digital portion of the 2920 can also perform other useful functions. For example, during a 9-bit A/D conversion, a 3-pole low-pass filter can be realized using the digital circuitry. The device's digital loop **(Fig 2)** includes the 2-port addressable RAM, with 40 words, a binary shifter and the ALU. Under program control, two RAM locations are simultaneously addressed from the 40 possible locations. The two 25-bit words are fetched, with the data from the A address passing through a binary shifter. The shifter allows scaling from 2^2 (a 2-bit left shift) to 2^{-13} (a 13-bit right shift). The scaled A value and the unscaled B value then propagate to the ALU. The ALU operates on these values with digital instructions specified by the program, loading the 25-bit result into the RAM's B-address location.

What makes the 2920 fast enough for real-time processing? Its analog operation, dual memory fetch, binary shift, ALU execution and write-back to RAM all occur in as little as 400 nsec, depending on clock rate (up to 10 MHz).

(a)



(b)

Mnemonics	Code/Condition	Operations	
Digital Instructions			
ADD		$(A \times 2^N) + B \rightarrow B$	
SUB		$B - (A \times 2^N) \rightarrow B$	
LDA [3]		$(A \times 2^N) + 0 \rightarrow B$	
XOR [3]		$(A \times 2^N) \oplus B \rightarrow B$	
AND		$(A \times 2^N) \cdot B \rightarrow B$	
ABS [3]		$\lvert(A \times 2^N)\rvert \rightarrow B$	
ABA		$\lvert(A \times 2^N)\rvert + B \rightarrow B$	
LIM		Sign (A)→ ± FS→B [4]	
ADD	CND( ) [2]	$(A \times 2^N) + B \rightarrow B$	IFF DAR(K) = 1
		$B \rightarrow B$	IFF DAR(K) = 0
SUB	CND( ) [2][7]	$B - (A \times 2^N) \rightarrow B$ & CY→DAR(K)	IFF $CY_P = 1$ [5]
		$B + (A \times 2^N) \rightarrow B$ & CY→DAR(K)	IFF $CY_P = 0$ [5]
LDA	CND( ) [2]	$(A \times 2^N) \rightarrow B$	IFF DAR(K) = 1
		$B \rightarrow B$	IFF DAR(K) = 0
ABA	CND( ) [8]	$(A \times 2^N) + B \rightarrow B$	
XOR	CND( ) [8]	$(A \times 2^N) \oplus B \rightarrow B$	
Analog Instructions			
IN(K)		Signal sample from input channel K	
OUT(K)		DAC to output channel K	
CVTS		Determine sign bit	
CVT(K)		Perform ADC on bit K	
EOP		Program counter to zero [6]	
NOP		No operation	
CND(K)		Select bit K for conditional instructions	
CNDS		Select sign bit for conditional Instructions	

NOTES:
1. Opcodes ALU and ADF are in binary notation; ADK is in decimal notation and represents the value K when appropriate.
2. CND( ) can be either CND(K) or CNDS, testing amplitude bits or the sign bit of the DAR respectively.
3. Clarification of CY_{OUT} sense for certain operations. For LDA, XOR, AND, ABS; $CY_{OUT} \rightarrow 0$.
4. B is set to full scale (FS) amplitude with the same sign as the A postoperand.
5. The previous carry bit (CY_P) is tested to determine the operation. The present carry bit (CY) is loaded into the Kth bit location of the DAR. Present carry (CY) is generated independent of overflow. It represents the carry (CY) of a calculated 28-bit result.
6. EOP also enables overflow correction if it was disabled during a program pass. The EOP must occur in ROM location 188.
7. For SUB CNDS operation, $\overline{CY} \rightarrow$ DAR(S).
8. Does not affect DAR. In this case, CND is used with XOR/ABA to enable/disable the ALU-overflow-saturation algorithm. Use of either instruction causes the ALU output to roll over rather than go to full scale with sign bit preserved. An EOP instruction also enables the ALU-overflow-saturation algorithm.

Fig 3—The assembler *for the 2920 uses the program format shown in* **(a)** *to specify the 24-bit word stored in the device's on-chip EPROM (note the combination of analog and digital instructions). The 2920's instruction set and operations are detailed in* **(b)**.

FIR filters with linear phase are easy to design

Short instruction set, unique format

The 2920 marries analog and digital operations by means of its instruction set. **Fig 3a** shows the program format used by the assembler to specify the 24-bit instruction word stored in the chip's EPROM. All processing subsystems are implemented using a combination of analog and digital instructions to input and output signals and/or data, and to realize the processing functions.

Analog input and output instructions are IN(K) and OUT(K), respectively. A sequence of IN(K) instructions, followed by the sign-conversion instruction (CVTS) and the amplitude-conversion instruction (CVT(K)), performs the input A/D conversion. A simple sequence of OUT(K) instructions is all that's needed to output a 9-bit amplitude on channel K.

Other analog instructions include EOP, which resets the program counter to zero after executing three more instructions, and NOP, which is simply no operation. Finally, CNDS or CND(K) are conditional operators that select and test a bit in the DAR for the conditional

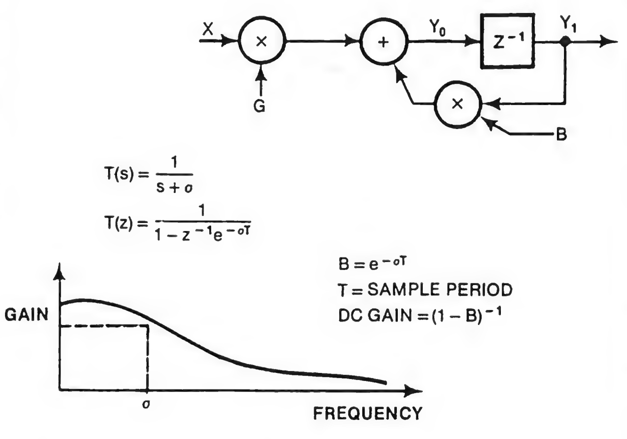


Fig 4—A sampled 1-pole-filter implementation *includes a unit delay (Z^{-1}) that's equivalent to one 2920 program pass.*

A brief review of digital filters

Digital filters are a class of networks that operate on discrete samples of a signal to achieve a desired transfer-function operation on that signal. They divide into two classes: *Nonrecursive filters* produce an output that's a function of only the previous and present inputs; *recursive filters* produce an output that's a function of both the past and present inputs and outputs.

The nonrecursive filter generates a finite impulse response and therefore is also termed an *FIR filter.* In contrast, the recursive filter, because of its feedback, has infinite impulse responses and is termed an *IIR filter.*

Recursive filters

The traditional approach to the design of IIR digital filters involves the transformation of an analog filter into a digital one meeting prescribed specifications. This is a reasonable approach for several reasons:

- The art of analog-filter design is highly advanced, and useful results can readily be achieved.
- Many analog-filter design methods have relatively simple closed-form formulas. Therefore, digital-filter synthesization techniques based on such analog formulas are simple to implement.
- In many applications, using a digital filter to simulate the performance of an analog linear time-invariant filter proves of interest.

Nonrecursive filters

FIR filters have several advantages:

- Units with exactly linear phase can be easily designed. This capability simplifies the approximation problem in many cases, when a designer is only interested in creating a filter that approximates an arbitrary magnitude response. Linear phase filters prove important in applications where frequency dispersion arising from nonlinear phase is harmful — e g, in speech-processing and data-transmission applications.
- Efficient realizations of FIR filters exist as both recursive and nonrecursive structures.
- FIR filters realized nonrecursively (i e, by direct convolution) are always stable.
- Round-off noise, which is inherent in realizations with finite-precision arithmetic, can easily be reduced to a small value for nonrecursive realizations of FIR filters.

FIR filters do, however, have several disadvantages, including the following:

- A large value of n (the number of filter taps) is required to adequately approximate sharp-cutoff filters. Hence, a large amount of processing is required to implement such filters when realized via slow convolution.
- The delay of linear-phase FIR filters need not always equal an integral number of samples, and this nonintegral delay can produce problems in some signal-processing applications.

ADD or LDA instructions, or define the carry bit's destination for the conditional SUB instruction.

The ALU arithmetic instructions (ADD, SUB and LDA) perform addition, subtraction and data transfer, respectively. When these instructions are conditioned, they can perform either multiplication or division by a variable, or data-dependent (conditional) switching. Other digital instructions include absolute value (ABS), absolute value and add (ABA) and ideal limit (LIM). **Fig 3b** details these commands.

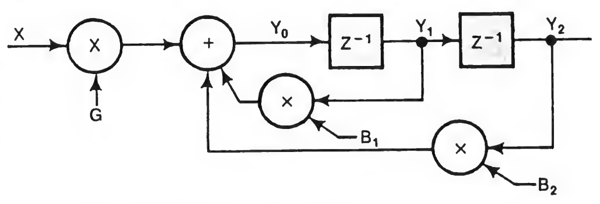
Digital-processing considerations

The very act of sampling a signal introduces a certain amount of distortion, termed aliasing noise. A tradeoff exists between input antialiasing-filter complexity and increased sampling rate for a given input-signal bandwidth. An ideal rectangular filter, for example, used at both the input and output of a sampled-data system, allows the sampling rate to be just twice the signal bandwidth. Practical filters, on the other hand, such as 5-pole Tchebyshev low-pass designs, can permit signal bandwidths of up to one-third the sampling rate; simpler filters require a greater ratio, depending on the overall system dynamic range needed.

Of course, digital processing of a sampled analog signal requires A/D conversion. The dynamic range of the input signal to be processed determines the number of bits of resolution needed: A rule of thumb is that one bit of A/D conversion yields 6 dB of dynamic range. Thus, a 9-bit conversion produces a 54-dB dynamic range (i e, the ratio of the maximum input signal to the minimum step size).

The ADC's finite resolution introduces the familiar quantization noise. But another, less familiar, form of this noise results from the use of finite word lengths in processing signals digitally. Typically, the number of bits per word needed to synthesize accurate digital filters increases as the filter bandwidth becomes smaller relative to a given sample rate. In this respect, the 2920 features 25-bit accuracy, plus the ability to expand to 28 bits during intermediate arithmetic operations; compare this capability with the eight or 16 bits possible in most digital processors. The net result is an ability to realize a broad range of filter bandwidths and sampling rates more accurately than most other processors.

(a)



$B_1 = 2e^{-\sigma T} \cos \omega T \cdot \quad B_2 = -e^{-2\sigma T}$

WHERE σ = REAL PART OF POLE

ω = IMAGINARY PART OF POLE

T = SAMPLE PERIOD

DC GAIN $= G/(1 - B_1 - B_2)$

MAX GAIN $= G/\left[(1 + B_2)\sqrt{1 + (B_1{}^2/4B_2)}\,\right]$

AT $f = \left(\frac{1}{2\sigma T}\right) \cos^{-1}\left(\frac{B_1(1 - B_2)}{4B_2}\right)$

$$T(s) = \frac{G}{s^2 + 2\sigma s + (\sigma^2 + \omega^2)},\quad T(z) = \frac{1}{1 - B_1 z^{-1} + B_2 z^{-2}}$$

(b)

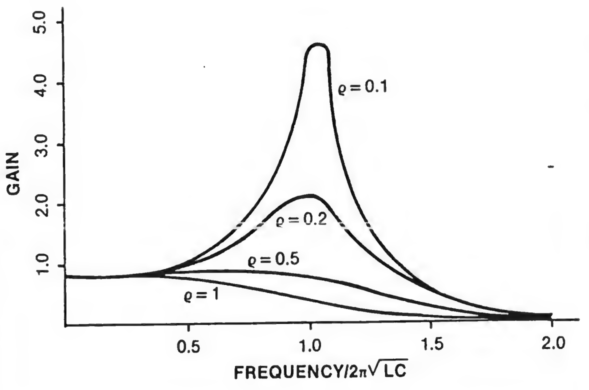


Fig 5—This realization of a complex-conjugate pole pair (a) *incorporates a loop within a loop. A section's gain is in* **(b)**.

The basic design approach

Conversion of an analog filter into a digital unit can be accomplished in several ways: impulse invariance, direct transformation using s-to-z-plane transform, and direct-synthesis techniques. In each approach, the known characteristics in the analog filter's frequency domain are converted to similar characteristics in the digital filter's z-plane.

Although the 2920 can implement both recursive (IIR) and nonrecursive (FIR) filters (see **box**), the primary goal of most design projects is to achieve an efficient implementation of a complex magnitude characteristic. Because IIR filters are most efficient for this job, the 2920 can achieve greater complexity using this type of digital filter.

Direct synthesis of digital filters to a desired amplitude characteristic is often the most efficient design technique when computer programs are available to aid this approach. The disadvantage of this method is that both software and high-speed computers are needed to perform the calculations.

Faced with this drawback, examine two design alternatives: the matched z transform and the bilinear transform. Both take advantage of analog-filter design techniques, then permit a simple conversion into a digital filter without the need for a computer. This article examines an implementation of the former.

Analog-filter-to-digital-filter transformation

Assuming that an analog filter has been designed that satisfies the needs of a particular application, how can this filter be implemented digitally? Given the analog design, the pole and zero locations in the analog s-plane can be derived from the filter's function. In

Pole/zero sample rate and locations must be known

general, the filter transfer function, T(s), can always be represented by a ratio of polynomials such as

$$T(s)=\frac{c_0 + c_1 s + c_2 s^2 + c_3 s^3 + \ldots + c_n s^n}{d_0 + d_1 s + d_2 s^2 + d_3 s^3 + \ldots + d_m s^m}$$

where $m \geq n$,

which can also be written as

$$T(s)=\frac{\sum_{i=0}^{n} c_i s^i}{\sum_{i=0}^{m} d_i s^i}=\frac{\text{zeros}}{\text{poles}}.$$

Factoring the polynomials yields the poles and zeros of the transfer function. Each complex pole (zero) pair can be represented by a quadratic function of the form $(s^2+k_1s+k_0)$, which has roots located where

$$S = -\frac{k_1}{2} \pm j\left(\tfrac{1}{2}\sqrt{4k_0 - k_1^2}\right).$$

Simple poles (zeros) can be represented by a function of the form $(s+k_0)$. And of course, these poles (zeros) can be plotted in the s-plane, where $s_i=\sigma_i \pm j\omega_i$; σ is the real part and $j\omega$ the imaginary part of the complex frequency. Each polynomial in the transfer function Y(s) can be represented by the product of quadratic functions and a simple function representing the roots of the polynomial. Therefore, T(s) can be written as the product of poles and zeros expressed in factored form.

Just as analog filters can be characterized by the locations of their poles and zeros and can be realized as a cascade of sections, each of which realizes a subset of poles and zeros, similarly, sampled filters can be characterized by their pole and zero locations. And as noted, simple methods exist for translating between a continuous filter and its sampled counterpart. The behavior of the sampled counterpart is similar to the original continuous filter's behavior at frequencies reasonably below half the sample rate.

Therefore, to design a filter using the 2920, you must first determine the sample rate and the locations of the filter's poles and zeros. In the examples that follow, it's assumed that the pole and zero locations for the equivalent continuous filter have been determined.

Given the list of poles and zeros, assign one filter section to realize each real pole and one to each

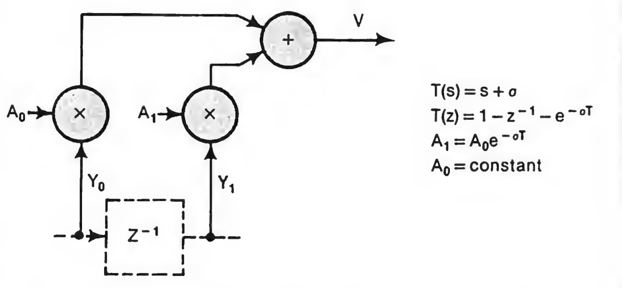


Fig 6—To implement a single (real) zero, *the elements depicted in solid lines must be added around delay elements, such as those shown in* **Figs 4** *and* **5.**

SINGULARITY	NORMALIZED (RAD/SEC)	DENORMALIZED (1 kHz)
SIMPLE POLE	$\sigma_0 =$ 0.83124 $\omega_0 =$ 0	$\sigma_0 = -5222$ RAD/SEC
COMPLEX-POLE PAIR	$\sigma_1 = -0.31128$ $\omega_1 = \pm 1.09399$	$\sigma_1 = -1955.8$ RAD/SEC $\omega_1 = \pm 6873.7$ RAD/SEC
COMPLEX-ZERO PAIR	$\sigma_2 =$ 0 $\omega_2 = \pm 2.2701$	$\sigma_2 =$ 0 RAD/SEC $\omega_2 =$ 14263 RAD/SEC

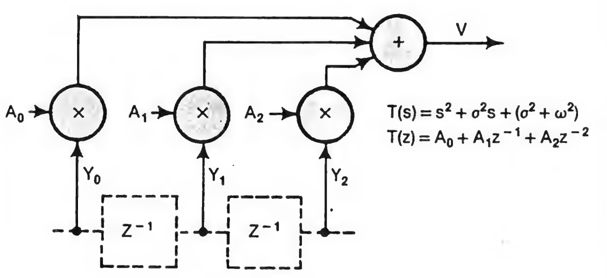

V
+
A_0
A_1
A_2
×
×
×
Y_0
Y_1
Y_2
z^{-1}
z^{-1}
$T(s) = s^2 + \sigma^2 s + (\sigma^2 + \omega^2)$
$T(z) = A_0 + A_1 z^{-1} + A_2 z^{-2}$

Fig 7—Pole-zero locations *for the low-pass filter described in the text are derived from the material in A I Zverev's* Handbook of Filter Synthesis *(Wiley & Sons, New York).*

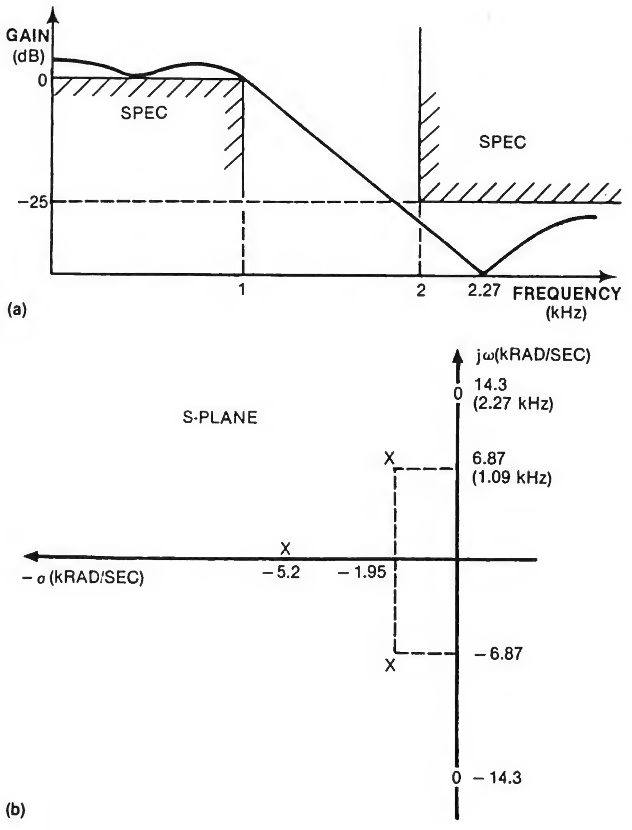


Fig 8—A gain-frequency plot (a) *and s-plane plot* **(b)** *are derived from the data in* **Fig 7.**

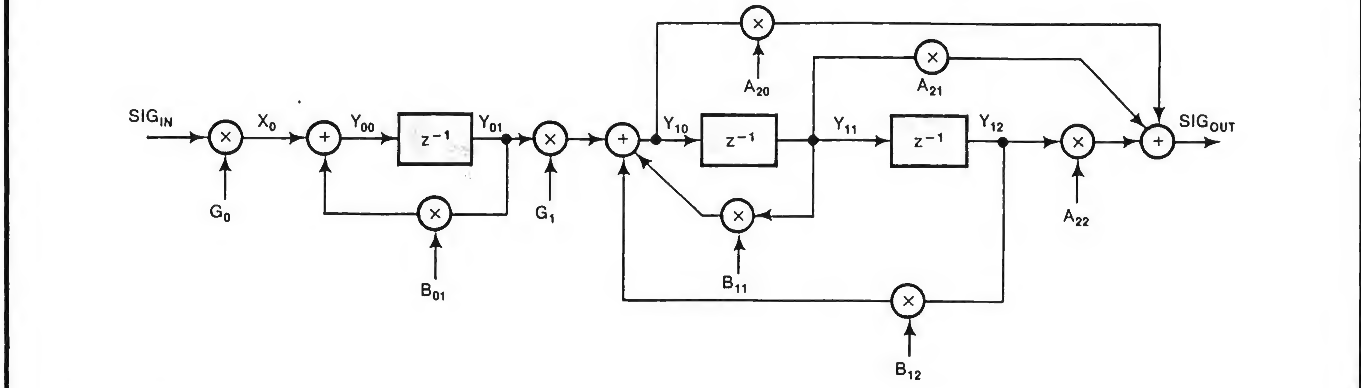


Fig 9—A cascaded digital-filter structure *requires less coefficient accuracy than a corresponding direct structure. The variable names shown define those used in the corresponding 2920 program* **(Fig 10).**

complex-conjugate pole pair; most zeros will be realized by adding them to one of the pole sections. Determine the gain of each section, then scale the inputs according to the design's needs. The following information describes the two basic types of filter section.

Simulating single real poles

Fig 4 shows a block diagram of a sampled 1-pole filter implementation. The block labeled Z^{-1} represents a unit delay; i e, a delay equivalent to one sample interval or one 2920 program pass. The blocks labeled × represent multiplications, in each case by a constant, and the block labeled + is an adder. The FORTRAN statements to implement **Fig 4** are as follows:

```
Y1=Y0
Y0=B*Y1+G*X
```

For the 2920, such FORTRAN statements must be converted to 2920 statements. For example, suppose you determine that B=0.9922 (B=0.11111110 in binary) and G=0.0078125 (G=0.0000010 in binary). The 2920 instructions could then be generated as follows:

OP	DEST	SOURCE	SHIFT	COMMENTS
LDA	Y1	Y0	R00	
LDA	Y0	Y1	R00	; Y0=1.0*Y1
SUB	Y0	Y1	R07	; Y0=B*Y1
ADD	Y0	X	R07	; Y0=B*Y1+G*X

Note that the comments show how the new value of Y0 is generated. In this case, the second instruction is superfluous and could be omitted.

Much of the design of such a filter section consists of determining the best values for B and G, consistent with design goals, yet easily realizable in 2920 code. Most of the 2920 support software functions to optimize 2920 code, subject to the appropriate design constraints. To illustrate the procedures involved, consider the following example:

Problem: *For a sample interval of 76.8 μsec, realize a single-pole filter with a time constant (1/σ) of 1.50 msec±1% and dc gain of 1.00±1%.*

The limits on B can be found from evaluating $B=e^{-\sigma T}$ for the range 1.485≤RC≤1.515; i e, 0.94960≤B≤0.95057. Expressed in binary, 0.1111001100011000 ≤B≤0.1111001101011000. The central value is B=0.95009 (0.1111001100111001 in binary).

Any value in the specified range can be chosen and still meet the design criteria. If you select a value of B=0.1111001101, it can be realized in five steps: $B=2^0-2^{-4}+2^{-6}-2^{-8}+2^{-10}$.

From the dc-gain equation, note that G=(1–B)±1%. Given the value for B just chosen, the range of acceptable values for G, expressed in binary arithmetic, is 0.000011001010<G<0.00001100111, with a target value of 0.0000110011. The target value can be realized, in four steps, as easily as any of the other values, and it can be expressed as $G=2^{-4}-2^{-6}+2^{-8}-2^{-10}$.

With the two constants (B and G) evaluated, you can directly convert them to 2920 assembly-language instructions by a method to be shown. However, before evaluating the final 2920 code, you should investigate overflow possibilities: By suitably limiting the input values, overflow can be made impossible. In other cases, a proper sequence of instructions can at least limit overflow to the last instruction, so that saturation occurs only if the final value is too large.

The following code sequence realizes this design problem's single-pole section. Here, terms have been ordered to prevent overflow from occurring on any but the last line. Comments show the contribution of instruction sequences.

OP	DEST	SOURCE	SHIFT	COMMENTS
LDA	Y1	Y0	R00	; Y1=Y0
LDA	Y0	X	R04	
SUB	Y0	X	R06	
ADD	Y0	X	R08	
SUB	Y0	X	R10	; Y0=G*X
SUB	Y0	Y1	R04	
ADD	Y0	Y1	R06	
SUB	Y0	Y1	R08	
ADD	Y0	Y1	R10	; Y0=G*X+(B-1)*Y1
ADD	Y0	Y1	R00	; Y0=G*X+B*Y1

Limiting input values helps avoid overflow

Simulating complex-conjugate pole pairs

Fig 5a shows a sampled realization of a complex pole pair with the 2920. Coefficients B_1 and B_2 control the frequency parameters; G adjusts the overall gain.

Fig 5b shows the frequency response of this type of stage. The designer's choice of parameter values determines both the frequency at which the gain peaks and the sharpness (Q) of the peak. The FORTRAN equations for a complex-conjugate pole-pair section are:

```
ISIS-II 2920 ASSEMBLER X102

ASSEMBLER INVOKED BY: AS2920 FILTER

Three Pole Two Zero Elliptical Low-pass Filter

  LINE  LOC OBJECT SOURCE STATEMENT

     1             $TITLE ('Three Pole Two Zero Elliptical Low-pass Filter')
     2
     3             Y22      EQU  Y11
     4             SIGOUT   EQU  Y22
     5
     6                 ;POLE 1
     7
     8    0 4008EF LDA Y11,Y10                  ; Y1=Y0
     9    1 40223E LDA Y10,DAR,R2               ; Y0=G0*X (INPUT SCALED DOWN BY 4)
    10    2 40001C ADD Y10,Y11,R1
    11    3 40007C ADD Y10,Y11,R4
    12    4 40009C ADD Y10,Y11,R5
    13    5 40003B SUB Y10,Y11,R10              ; Y10= G0*X + B*Y11
    14
    15                 ;POLE 2 & 3
    16
    17    6 4200EF LDA Y22,Y21                  ; Y22 = Y21
    18    7 4608EF LDA Y21,Y20                  ; Y21 = Y20
    19    8 44085E LDA Y20,Y10,R3               ; Y20 = G1*Y10 (STAGE PROPAGATION SCALED DOWN BY 8)
    20    9 4600BC ADD Y20,Y21,R6
    21   10 4600FD ADD Y20,Y21,R0
    22   11 46003C ADD Y20,Y21,R2               ; Y20 = B1*Y21
    23   12 44001A SUB Y20,Y22,R1
    24   13 44005A SUB Y20,Y22,R3
    25   14 44009A SUB Y20,Y22,R5
    26   15 4400BA SUB Y20,Y22,R6               ; Y20 = B1*Y1 + B2*Y2 + G1*Y10
    27
    28                 ;ZERO 1 & 2
    29
    30   16 4208ED ADD SIGOUT,Y20               ; SIGOUT = A0*Y20 + A2*Y2
    31   17 42002A SUB SIGOUT,Y21,R2
    32   18 42008A SUB SIGOUT,Y21,R5
    33   19 4200EA SUB SIGOUT,Y21,R8            ; SIGOUT = A0*Y0 + A1*Y1 + A2*Y2
    34
    35
    36   20 4044CF LDA DAR,SIGOUT,L1            ; OUTPUT SCALED UP BY 2
    37
    38
    39             END

SYMBOL:                       VALUE:

Y22                                0
Y11                                0
SIGOUT                             0
Y10                                1
Y21                                2
Y20                                3

ASSEMBLY COMPLETE


ISIS-II 2920 ASSEMBLER X102

Three Pole Two Zero Elliptical Low-pass Filter

ERRORS    =    0
WARNINGS  =    0
RAMSIZE   =    4
ROMSIZE   =   21
```

Fig 10—The filter-program listing *for the low-pass design example shows only the 2920's digital instructions. However, the analog and digital instructions can be written in their respective fields in the same line of machine code; the chip executes both in parallel.*

```
Y2=Y1
Y1=Y0
Y0=B1*Y1+B2*Y2+G*X
```

Once you find the coefficients of the third equation, you can convert all the equations to 2920 code using the procedures previously described. Thus, the major portion of the design consists of finding those values for the coefficients that meet the design requirements, yet require the minimum number of 2920 steps to realize.

Problem: *For a sample interval of 76.8 μsec, synthesize a section with resonance at 1000 Hz±0.5% and Q in the range 75≤Q≤100. The peak gain should be 1.0±10%.*

A complex-conjugate pair of s-plane poles at $s=-a\pm jb$ has an impulse response which rings at a frequency $f=b/2\pi$ and a value for Q given by $Q=b/2a$.

Thus, $bT=0.48255\pm0.0024$ and, at $bT=0.48255$, aT falls in the range $0.002412\leq aT\leq 0.003217$. Expressing the negative of B_2 in binary form gives:

$$0.111111001011 \leq -B_2 \leq 0.111111101100.$$

A value that falls in this range and can be expressed in only three powers of 2 is

$$-B_2=0.111111101=2^0-2^{-7}+2^{-9}=0.99414.$$

Having established B_2, you can find B_1 using the relationships $e^{-aT}=\sqrt{-B_2}$ and $B_1=2e^{-aT}\cos(bT)$. In binary notation, $1.1100010011\leq B_1\leq 1.1100001110$. A suitable value is $B_1=1.110001=2^{-1}-2^{-2}+2^{-6}=1.7656$.

To test the values of B_1 and B_2 chosen, calculate from the equations in **Fig 5** the resonant frequency ($f_r=1001.8$) and Q (Q=82); maximum gain is then

$$G_m = \frac{G}{0.002724} \text{ at } f_m = 1001.8.$$

To meet the constraints, $G=2^{-8}-2^{-10}=0.00293$ is adequate.

You can write corresponding 2920 code by evaluating the coefficients:

OP	DEST	SOURCE	SHIFT	COMMENTS
LDA	Y2	Y1	R00	; Y2=Y1
LDA	Y1	Y0	R00	; Y1=Y0
LDA	Y0	Y1	L01	
SUB	Y0	Y1	R02	
ADD	Y0	Y1	R06	; Y0=B1*Y1
SUB	Y0	Y2	R00	
ADD	Y0	Y2	R07	
SUB	Y0	Y2	R09	; Y0=B1*Y1+B2*Y2
ADD	Y0	X	R03	
SUB	Y0	X	R10	; Y0=B1*Y1+B2*Y2+G*X

If the inputs are scaled so that overflows might occur in the calculation of Y_0, a reordering of the terms could prove necessary. In this particular design example, at the third step, Y0=2*Y1—a value that would produce overflow if $|Y_1|>0.5$. Reordering the steps to add the 2*Y1 term last might reduce the probability of overflow. Alternatively, you could reduce the gain at the filter input and boost the filter output to compensate.

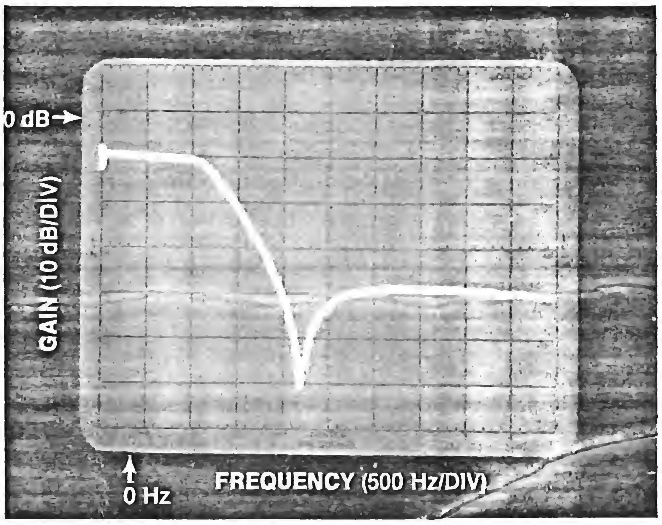


Fig 11—Frequency response *of the 3-pole, 2-zero low-pass elliptical filter programmed in* **Fig 10** *illustrates the performance achieved by the 2920.*

Realizing zeros in basic filter sections

The building blocks described thus far realize only poles (i e, the zeros are all at infinity). To introduce zeros at other locations, you can add additional elements to the sections already described. Each section remains unchanged except for the addition of blocks that combine some of the Y values to generate new outputs.

For example, **Fig 6** shows the blocks that can be used to add a single (real) zero. The FORTRAN equation of a zero realization is

```
V=A0*Y0=A1*Y1.
```

To produce a zero equivalent to a continuous zero at $s=-a$, the values for A_0 and A_1 must have the relationship

$$A_1=A_0\,(e^{-\sigma T}),$$

where the value for A_0 is arbitrary. For a zero at dc, $A_1=A_0$.

Complex-conjugate zero pairs

To realize complex-conjugate pairs of zeros, the three values Y_0, Y_1 and Y_2 must be available. If they are not available as part of a basic complex-conjugate pole stage, you can add additional delay stages. The structure depicted in **Fig 7** realizes the equivalent of a complex-conjugate pair of zeros at $s=\sigma\pm j\omega$ via the FORTRAN equation

```
V=A0*Y0+A1*Y1+A2*Y2.
```

A_0, A_1 and A_2 must meet the following requirements, where the value of A_0 is arbitrary:

$$A_1=2A_0e^{-\sigma T}\cos\omega t$$
$$A_2=A_0e^{-2\sigma T}.$$

Designing a complex low-pass filter

All of the individual sections discussed thus far can be added together to solve real-world applications.

Problem: *For a multifrequency receiver, design a low-pass filter with a passband from dc to 1 kHz and less than 1 dB of ripple. Rejection for frequencies above 2 kHz must exceed 25 dB.*

Complex-conjugate zero pairs require values for Y_0, Y_1 and Y_2

A study of filter curves (such as the nomographs in A I Zverev's *Handbook of Filter Synthesis*, Wiley & Sons, New York, pgs 140-143) shows that an elliptic-function filter with three poles, two zeros and a 25% reflection coefficient can meet these requirements. Pg 178 of Zverev (for $\theta=30$) shows the filter pole/zero values normalized to 1 rad/sec bandwidth. The normalized and denormalized values are listed in **Fig 7** for the selected filter. **Fig** 8 shows the corresponding gain-frequency and s-plane plots.

Once the required poles and zeros have been identified, the basic block diagram of the digital filter can be drawn and the coefficients calculated. (The three poles require three delay elements, two of which can be used to implement the two zeros.) The cascaded structure shown in **Fig** 9 not only simplifies these calculations, but also realizes a digital-filter structure that requires less coefficient accuracy than a direct (uncascaded) implementation would.

For the purpose of this design example, assume a sample rate of 10 kHz (a period of 100 μsec). In addition, assume that coefficient accuracies of ±1% or better are required. Then the required calculations can be performed as follows:

Simple-pole calculations

$B_{01} = e^{-\sigma T} = \exp[-(5222)(0.0001)]$
$= 0.593214.$

$B_{01}+1\% = 0.10011001011$ (in binary)
$B_{01} = 0.10010111110.$
$B_{01}-1\% = 0.10010110010.$

A value in this range can be represented as
$= 2^{-1}+2^{-4}+2^{-5}-2^{-10}.$

dc gain $= 1/1-B_{01} = 2.4583.$

Complex-pole calculations

$B_{11} = 2e^{-\sigma T}\cos\omega T$
$= 2\exp[-(1955.8)(0.0001)]$
$\cos[(6873.7)(0.0001)]$
$= 1.271229.$

$B_{11}+1\% = 1.01001000101$ (in binary)
$B_{11} = 1.01000101011$
$= 2^0+2^{-2}+2^{-6}.$
$B_{11}-1\% = 1.01000010001.$

$B_{12} = -e^{-2\sigma T} = \exp[-(2)(1955.8)(0.0001)]$
$= -0.67627194.$

$B_{12}+1\% = 0.10101110110.$
$B_{12} = 0.10101101001.$
$B_{12}-1\% = 0.10101011011.$
$= -2^{-1}-2^{-3}-2^{-5}-2^{-6}.$

max gain $= [(1+B_2)\sqrt{1+(B_1^2/4B_2)}]^{-1}$
$= 4.7949.$

Complex-zero calculations

$A_{10} = 1.$

$A_{11} = -(2)(A_{01})\, e^{-\sigma_2 T}\cos\omega_2 T$
$= -0.28798805$
$= 0.01001001101$ (in binary)
$= -2^{-2}-2^{-5}-2^{-8}.$

$A_{12} = A_{01}e^{-2\sigma_2 T}$
$= (1)\exp(0)$
$= 1.0$

Fig 10 shows the 2920 program that implements these results, while **Fig 11**'s scope photo depicts the actual performance achieved. **EDN**

References

1. *2920 Design Manual,* Intel Corp, 3065 Bowers Ave, Santa Clara, CA 95051.

2. "Single-chip NMOS Microcomputer processes signals in real time," *Electronics,* March 1, 1979, pg 105.

3. "First single-chip signal processor simplifies analog design problems," *Electronic Design,* September 27, 1979, pg 50.

4. "Program a spectrum analyzer on a 1-chip real-time signal processor," *Electronic Design,* November 8, 1979.

Author's biography

Robert E Holm received his BSME/EE degree from Arizona State University in 1965 and his MSEE from the University of Santa Clara in 1968. In 1965, he joined IBM's San Jose, CA product-test laboratory, then moved to Electromagnetics Systems Labs, Sunnyvale, CA, in 1966 as a design engineer. After joining ArgoSystems, Sunnyvale, CA, in 1970 as a senior member of the firm's technical staff, he managed programs that developed state-of-the-art signal-processing algorithms and equipment. Since joining Intel in December 1977, Bob has held the positions of product manager for the 2920 signal processor and manager of applications engineering; he is currently product marketing manager for telecommunications.

intel®

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, CA 95051 (408) 987-8080

Printed in USA/ K59/ 0180/ 25K/ BL AFN 01308A